

IGNIS Edge AI Gateway: An Evidence-Aware Architecture, Prototype Evaluation, and Integration Roadmap

Aaron Singh

Abstract—This paper presents IGNIS Edge AI Gateway, a project in a twenty-project AI infrastructure portfolio. The current repository is an active prototype with implemented behavior, automated correctness tests, and explicit boundaries around unverified hardware or production claims. We describe the system model, current implementation, goals, and evaluation method, then propose five integrations that advance the project toward reproducible system-level validation. A local audit executed 3 project tests successfully. No accelerator, deployment, or performance conclusion is inferred unless a corresponding committed benchmark or profiler artifact exists.

Index Terms—IGNIS, AI infrastructure, reproducibility, prototype validation, systems evaluation, hardware–software co-design

I. INTRODUCTION

IGNIS Edge AI Gateway (P13) addresses a bounded problem within the portfolio’s track-04-robotics-edge track. Its declared status is *Active Prototype* [1]. The project validates its central mechanism before adding device-specific acceleration, production orchestration, or real-world hardware. This ordering matters because optimization without a trusted reference can make incorrect behavior appear successful.

The project goals are to establish deterministic behavior, encode correctness as tests, create machine-readable evidence, identify limiting resources through measurement, and integrate results with adjacent portfolio systems without losing provenance. The repository contains one paper named exactly after its singular folder: P13-ignis-edge-ai-gateway.tex.

II. TECHNICAL CONTEXT

Edge and robotic systems operate under resource, latency, privacy, and safety constraints. Development should progress from deterministic simulation to representative sensors, middleware, and hardware-in-the-loop validation. The portfolio benchmark standard requires environment, workload, method, metric, artifact, reproduction, and limitation fields; unknown values remain explicitly unmeasured [3].

III. SYSTEM MODEL AND ARCHITECTURE

The prototype is organized around the following domain model:

$$r(j) = \begin{cases} \text{edge,} & \text{private}(j) \\ \text{edge,} & d_j \leq c_e \wedge L_e \leq L_j \\ \text{cluster,} & \text{otherwise} \end{cases} \quad (1)$$

The equation is a design and test abstraction rather than a claimed empirical law. It supports invariants and expected-value checks while later implementations replace synthetic inputs with representative workloads or devices.

The software architecture contains an input/configuration layer, a deterministic core, validation and evidence output, and a local visualization. The principal inspected source artifacts are `python/gateway.py`. Unsupported real-world conditions are surfaced as limitations rather than silently simulated.

IV. DETAILED SCRIPT OPERATION AND RATIONALE

The gateway validates job fields and applies explainable edge-versus-cluster routing. Privacy-sensitive jobs stay local, while size and latency constraints guide other placements. The script proves policy behavior without implying that a device, API, authentication layer, or network is connected.

The execution path is:

- 1) Validate the incoming job schema and numeric constraints.
- 2) Apply the privacy-local rule first.
- 3) Check whether edge capacity and latency satisfy the job.
- 4) Route remaining work to the cluster with an explanation.
- 5) Return a deterministic decision suitable for API and device adapters.

V. IMPLEMENTED PROTOTYPE

The metadata and source audit found these completed features [2]:

- Validated Python job schema
- Explainable edge/cluster routing
- Privacy-local rule and tests

`python3 -m unittest discover -s tests -p 'test_*.py'` completed successfully with 3 tests. The inspected test artifacts are `tests/test_gateway.py`. This is evidence of local correctness for encoded cases, not production scale or hardware performance.

VI. TESTING METHODOLOGY AND OBSERVED RESULTS

Testing uses Python’s standard `unittest` discovery and exercises the public behavior of the reference implementation. The audit reran the suite from the project folder with `python3 -m unittest discover -s tests -p 'test_*.py'`. All 3 discovered tests passed. The result establishes correctness only for the encoded

TABLE I
IMPLEMENTATION ARTIFACTS AND WHY THEY EXIST

Artifact	Observed responsibility	Engineering rationale
python/gateway.py	Job, validate, route, main	Implements the inspectable, unit-tested project core.
scripts/reproduce.sh	Fixed test and demonstration entry point	Gives another developer one command for local reproduction.
PROJECT.yaml	and	Separates declared intent from evidence-backed implementation.
ANALYSIS.md		Makes outputs inspectable without upgrading simulation into a hardware claim.
streamlit_app.py		

local cases; it does not establish accelerator correctness, real-device behavior, production reliability, or benchmark completion.

No numerical performance result is promoted by this test run. Where scripts emit JSON or JSONL, those outputs remain raw or simulation-specific until a reviewed summary includes hardware, software, workload, warm-up, repetition, correctness threshold, Git revision, and limitations.

VII. CLAIM BOUNDARIES AND RISKS

The project records these unverified or excluded claims:

- No real edge device or network is connected

The main risk is confusing synthetic or modeled behavior with deployed-system behavior. Other risks include incomplete workloads, platform-dependent timing, missing failure injection, and interfaces not yet exercised across device boundaries. Performance claims require a reviewed record meeting the portfolio standard.

VIII. EVALUATION PLAN

Evaluation proceeds through correctness tests, deterministic reproduction with Git and environment metadata, repeated benchmarks reporting latency/throughput/memory/error metrics, and a named profiler capture tied to exact hardware and source revision. Success requires reference equivalence within a documented tolerance, preservation of safety and resource invariants, clear failures, and evidence reproducible from a clean environment.

IX. GOALS, MILESTONES, AND SUCCESS CRITERIA

The project goals are staged so that correctness precedes performance and integration. A goal is complete only when its proof artifact is committed or otherwise reviewable; prose or a simulated number alone is insufficient.

X. FUTURE WORK AND INTEGRATIONS

The five project-specific next steps are:

- 1) FastAPI endpoint
- 2) Retries and authentication
- 3) Real device adapter
- 4) Add signed manifests, local buffering, and offline recovery.
- 5) Integrate placement with P16 and device health with P12.

The early items complete declared evidence; the later items connect downstream portfolio consumers. Each integration should add tests and a reviewable artifact such as JSONL evidence, a report, profiler capture, deployment manifest, trace, or labeled data set.

XI. CONCLUSION

IGNIS Edge AI Gateway is an evidence-aware active prototype: its implemented behavior and tests are real, while unbuilt hardware, deployment, and performance goals remain labeled. Completing the five integrations in dependency order will advance it from a learning artifact toward a credible portfolio component.

REFERENCES

- [1] Aaron Singh, "IGNIS Edge AI Gateway PROJECT.yaml," local portfolio repository.
- [2] Aaron Singh, "IGNIS Edge AI Gateway: README, ANALYSIS, source, and test artifacts," local portfolio repository.
- [3] Aaron Singh, "AI Infrastructure Portfolio Benchmark Standard," local portfolio repository.

TABLE II
AUDITED TEST MATRIX

Test artifact and case	Behavior being checked	Observed result
tests/test_gateway.py:test_private_local	Private local.	Pass (local)
tests/test_gateway.py:test_large_cluster	Large cluster.	Pass (local)
tests/test_gateway.py:test_invalid	Invalid.	Pass (local)

TABLE III
DECLARED STATUS VERSUS AUDITED EVIDENCE

Audit field	Finding	Evidence source
Declared status	Active Prototype	PROJECT.yaml
Evidence-backed status	Active local prototype; 3 tests passed	Source plus local unittest run
Accepted measured results	None recorded in measured_results	PROJECT.yaml
Mismatch / claim boundary	No real edge device or network is connected	PROJECT.yaml and ANALYSIS.md
Next proof required	FastAPI endpoint; Retries and authentication	Planned features

TABLE IV
PROJECT GOALS AND REQUIRED PROOF

ID	Goal	Completion evidence
G1	FastAPI endpoint	Passing tests and a reviewed source artifact
G2	Retries and authentication	Machine-readable result with reproduction metadata
G3	Real device adapter	Reference-equivalence or domain-correctness report
G4	Expose an authenticated retry-safe API	Named profiler, deployment, or integration artifact
G5	Connect a real edge device adapter	Dashboard/report link preserving provenance and limitations